A Reinforcement Learning Approach to Economic Dispatch using Neural Networks

E.A.Jasmin, T.P.Imthias Ahamed and V.P.Jagathiraj

Abstract: **This paper presents a Reinforcement Learning (RL) approach to economic dispatch (ED) using Radial Basis Function neural network. We formulate the ED as an N stage decision making problem. We propose a novel architecture to store Q-values and present a learning algorithm to learn the weights of the neural network. Even though many stochastic search techniques like simulated annealing, genetic algorithm and evolutionary programming have been applied to ED, they require searching for the optimal solution for each load demand. Also they find limitation in handling stochastic cost functions. In our approach once we learn the Q-values, we can find the dispatch for any load demand. We have recently proposed a RL approach to ED. In that approach, we could find only the optimum dispatch for a set of specified discrete values of power demand. The performance of the proposed algorithm is validated by taking IEEE 6 bus system, considering transmission losses.**

I. INTRODUCTION

A Power system consists of a large number of units having a variety of characteristics. The load demand is not at all constant; it varies from time to time. Meeting this varying load with minimum cost of generation, while satisfying all the constraints associated with the system is an important problem [1]. For solving this constrained optimization problem, termed as Economic Dispatch, various techniques have been developed.

The conventional methods include lambda iteration [1], base point participation factor [2] etc. These methods are not suitable when the fuel cost functions are non convex. Dynamic Programming is a good solution for non convex cost functions [1]. But it suffers from curse of dimensionality. A variety of stochastic search techniques including Genetic Algorithm, Simulated Annealing, Partition Approach Algorithm, Evolutionary Programming [3-6] etc. have been proposed by different researchers. But these techniques require searching for the optimal solution for each load demand. Typically, we have to repeatedly obtain the solution for various load demands. Hence if we want to find the dispatch for 24 different load values we have to run the stochastic algorithm 24 times. Moreover, in practical situations the cost of generation may be stochastic. This stochastic cost functions are to be handled by Economic Dispatch algorithms. Most of the existing algorithms assume deterministic cost function. However, Reinforcement Learning algorithm can handle stochastic cost function in a straight forward manner. Our goal is to develop economic dispatch algorithm using data available from the power industry.

Recently we have proposed Reinforcement Learning (RL) based approach to Economic Dispatch problem [12]. We denote this algorithm as RLED. RLED involves learning of the so called Q – values. Q values are defined for state-action pairs (x, a). Q (x, a) denotes how good it is to take action 'a' in state 'x'. In the context of Economic Dispatch power to be dispatch is the state and the allocation of power to each unit is the action. In RLED, we assumed power demands will take only from a set of discrete values, and allocation of power to each unit also can take only discrete values. That is, number of states and actions are finite. Hence, we could store Q-values in a lookup table (matrix).

In RLED, once we learn the Q – values, we can find the dispatch for any specified discrete values of power demand almost instantaneously. Hence the time taken for finding the dispatch for one load or 10 different loads was almost the same. One limitation of RLED is, it involves quantization of power demands. For example, if the minimum and maximum power demand possible for a power system is D^{min} and D^{max} respectively, we have to quantize the power demand to finite values D^{min}, $D^{min}+S$, $D^{min}+2S$,……D^{max}. Hence using RLED it is only possible to find the dispatch corresponding to these values. However, this problem can be over come by choosing a small step size "S". But choosing a small step size makes the algorithm inefficient.

In this paper, we propose the use of function approximation to store the Q – values [7, 8] and present the algorithm to find the dispatch for any values of power demand from D^{min} to D^{max}. It may be noted that applications of RL to power system have been few [9-11]. We hope that this paper will generate more interest in application of RL to power systems.

The rest of the paper is organized as follows. Mathematical formulation of Economic Dispatch problem is explained in next section. In section III, we explain the Multi stage decision making problem and Reinforcement Learning method of solution. In section IV, Economic Dispatch is formulated as MDP and the RLED solution is discussed. Architecture of Radial Basis Function Networks used for solution is given in Section V. The Algorithm developed for solution of Economic Dispatch neglecting transmission losses is given in Section VI and the algorithm is extended in section VII to account the transmission losses also. Simulation studies are given in Section VIII. Conclusions are given in the last section.

II. ECONOMIC DISPATCH

Consider a power system having N generating units. Let P_T be the power demand to be satisfied with these N units at any slot of time and let P_L be the total transmission loss in the system. Economic Dispatch is to find an optimum schedule of

E.A.Jasmin,Govt Engg.College,Thrissur,Kerala, eajasmin@gmail.com
Dr. T.P.Imthias Ahamed, Dept. of Electrical& Electronics, T.K.M. College of Engineering, Kollam,Kerala email : imthiasa@gmail.com
Dr.V.P.Jagathiraj, Professor, School of Management Studies, Cochin University of Science and Technology,Kerala email: jagathi@cusat.ac.in

power allocation among these *N* units. The allotment should be in such a way that the cost of generation should be minimum. At the same time, generating unit power constraints should also be met.

The objective function of Economic Dispatch problem is the total cost for supplying the demanded load C_T. The problem is to minimize

$$C_T = \sum_{i=1}^{i=N} C_i(P_i) \quad (1)$$

$$\textbf{s.t:} \quad \Sigma(P_i) - P_T - P_L = 0 \quad (2)$$

$$P_i^{min} \le P_i \le P_i^{max} \text{ for } i = 1 \text{ to } N \quad (3)$$

III. MULTI STAGE DECISION MAKING PROBLEM AND REINFORCEMENT LEARNING

To make the paper self contained and introduce notations used let us consider an N stage decision making problem. Let the system be in state x_0 in stage 0. If we take an action a_0 system will move to state x_1. When the system moves from x_0 to x_1, it will incur a cost $g(x_0, a_0, x_1)$. In general, in the k^{th} stage, let the system be in state x_k. If we take an action a_k, system will move to state x_{k+1} and will incur a cost of $g(x_k, a_k, x_{K+1})$. For an N stage MDP, system will reach an absorption state in stage N.

MDP is the problem of finding actions $a_0, a_1, a_2, \ldots..a_{N-1}$, such that the total expected cost $E\left[\sum_{k=0}^{k=N-1} g(x_k, a_k, x_{k+1})\right]$ is minimised. In general, the cost $g(x_k, a_k, x_{k+1})$ could be a random variable.

When the system is in state *x*, we will take an action *a* based on some rule or "policy". In RL literature, the policy is denoted by $\Pi(\ )$. Thus, if we are following a policy $\Pi(\ )$, the action taken in state *x* is $\Pi(x)$. We can think of Π as a mapping

$$\Pi : \chi \rightarrow \mathcal{A}$$

where χ is the state space and $\mathcal{A}$ is the action space. In RL literature [7, 8] there are several algorithms to find optimal policy. Here, we explain one such algorithm, the Q learning algorithm.

Q learning algorithm involves learning the Q values for all state action pairs. Q value for a state action pair (x, a) is defined as the total expected cost, if we start from state ***x***, take an action ***a*** thereafter follow the optimal policy.

$Q(x,a) = E\left[\sum_{k=0}^{N-1} g(x_k, a_k, x_{k+1})\right]$,

$x_0 = x, a_0 = a\ a_k = \pi^*(x_k)$Suppose we know the Q values for all actions possible in state x'. That is, suppose we know $Q(x', a_1), Q(x', a_2), \ldots\ldots\ldots\ldots\ldots Q(x', a_m)$ where $\{a_1, a_2, \ldots\ldots\ldots a_m\}$ are the possible actions in state x'. Then we can find the optimal policy or best action at state x' as

$\Pi(x') = argmin_{a'} Q(x', a')$. Thus by learning Q–values for different possible state – action pairs, we can find the optimal actions.

IV. ECONOMIC DISPATCH AS A MILTI STAGE DECION MAKING PROBLEM AND SOLUTION THROUGH RLED

In this section, we show how ED can be modelled as a MDP. We also briefly present RLED. Consider a power system with *N* generators $G_1, G_2, \ldots\ldots\ldots G_N$ which has to share a total load P_T. This problem is treated as an N stage problem. At each stage of the MDP, a decision is made which correspond to an allocation to one among the N units. In order to apply the RL strategy, we should identify the state space and action space at each of these N stages.

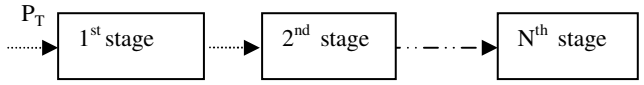

Fig. 1 MULTI STAGE PROBLEM STRUCTURE

The state of the system at each stage *i*, is denoted as D_i, where D_i is the power to be scheduled among $N-1-i$ generating units. That is, in the first stage state $D_1 = P_T$ is the power to be scheduled among the N generating units.

The state space at stage *i* is limited by the minimum and maximum values $D_i^{min} and\ D_i^{max}$. When an action or allocation is selected at stage *i*, it should satisfy basically two conditions. First, the remaining *N-1-i* Units should have at least their minimum generation levels to be allocated since they are already decided to be on line. Secondly, all these *N-1-i* units can generate only up to their maximum limit. Thus the possible states at each stage *i* is decided by the minimum and maximum amount of power that can be generated by the *N–1–i* generating units. Therefore,

$$D_i^{min} = \sum_{k=i}^{k=N} P_i^{min}; \qquad D_i^{max} = \sum_{k=i}^{k=N} P_i^{max} \quad (4)$$

When we choose step size as S_i *MW*, then there will be $m_i = (D_i^{max} - D_i^{min}) / S_i$ different possible states for state D_i. State space at stage i, $\chi_i = \{D_i^{min}, D_i^{min} + S_i, \ldots\ldots\ldots D_i^{max}\}$

In general at any stage *i*, system will be in any one of the possible states $D_i \in \chi_i$. The next step in the MDP is to apply an action a_i^k from the permissible action set $\mathcal{A}_i$. For the same, we identify the set of permissible actions at each stage of the system. In this MDP, we take an action as allocation of power. At stage *i*, action a_i^k is one of the possible allocations to i^{th} unit. For making the action set finite, we discretise the action space. If S_a is the step size chosen for the action set, then at any stage *i*, the action set $\mathcal{A}_i \in \{P_i^{min}, P_i^{min} + S_a, \ldots P_i^{max}\}$.The maximum number of possible actions will *be* $n_i = (P_i^{max} - P_i^{min}) / S_a$.

Then the learning proceeds as follows. The state of the system at the first stage is $D_1 = P_T$ (note that $P_T = P_1 + P_2 + \ldots\ldots\ldots P_N$)*. In the first stage, system choose an action $a_1^k \in \mathcal{A}_1$ and hence the power allocation to first unit $P_1 = a_1^k$ is decided. On taking an action the system moves to the next stage with state as D_2 and incur a cost of $g(D_1, a_1^k, D_2)$. Here $D_2 = D_1 - P_1$ and $g(D_1, a_1^k, D_2) = C_1(P_1)$, cost of generating P_1 units of power by generator G_1. In the second stage, decision making problem is similar. We have *N -1* generating units, $G_2, G_3, \ldots\ldots\ldots G_N$. These generators together should supply D_2 units of power. In the language of MDP, the new state is D_2 and the decision maker has to take an action $a_2^k \in \mathcal{A}_2$ and system will move to D_3 with an incurred cost of $g(D_2, a_2^k, D_3) = C_2(P_2)$ (Where $P_2 = a_2^k$) This proceeds until all the allocations ($P_1, P_2, \ldots\ldots P_N$) are over.

* In our notation, $P_1, P_2, \ldots\ldots\ldots P_N$ etc. are variables.

When χ_i and $\mathcal{A}_i$ are treated as finite or having discretised values as explained above, we have a finite number of state – action pairs at each stage. Then the Q values can be stored in a table and during the policy retrieval phase, using the look up table best action a_i (having minimum Q value) corresponding to any state D_i can be obtained. Here we consider D_i values as continuous in the range D_i^{min} to D_i^{max} . Since the input space is continuous, it is not possible to store Q values in a table. So we represent Q–values using a parameterised class of functions $\{Q(D_i, a_i, \theta): \theta \in \Re^l\}$, where θ is a parameter vector. Now learning optimal Q values involve learning the optimal parameter vector θ^* such that $Q(D_i, a_i, \theta^*)$ is a good approximation of $Q^*(D_i, a_i)$.

There are many parameterized classes of functions that one can consider while looking for an approximating function. Here, we use Radial Basis Function (RBF) neural network [13,14].

In most applications, where neural networks are used for function approximation, one requires a set of training data. In our context, we require training set of the form $\{< (D_i, a_i); Q^*(D_i, a_i) >; i=1, 2,.....N\}$. However, we do not know $Q^*(D_i, a_i)$ for any (D_i, a_i). Thus, we cannot use any standard supervised learning algorithm. Here we develop a Reinforcement Learning algorithm for training the RBF network.

V. ARCHITECTURE OF THE RBF NETWORK

Next step is to choose a suitable architecture of the RBF network in order to store the Q – values. We model N unit power system, as N stage decision making problem. At each stage we are having state space χ_i and action space $\mathcal{A}_i$. In order to store the Q values we use RBF networks. In this N stage problem we need N RBF networks. RBF network of each stage consists of an input layer, hidden layer and an output layer. The input layer is made up of a single node for connecting the input state variable (D_i) to the network. As explained earlier D_i can take any of the possible values in the range D_i^{min} to D_i^{max} . With a chosen value of diascretisation step S, the hidden layer consists of m_i hidden nodes. This layer applies a non linear transformation from the input space to the 'm_i' dimensional hidden space, $\Re^{mi}$. Number of output nodes is decided by the number of actions since each output represents a Q value, $Q(D_i, a_i^k)$. Therefore output layer consists of n_i linear nodes (the maximum number of possible actions decided by P_i^{min} and P_i^{max} and step size S_a) which combine the outputs of hidden nodes .

The architecture of the RBF network for the first stage is given in fig (2). The input to this RBF network is D_1, the total power to be scheduled among generators G_1, G_2, ...G_N .The output of the network is $Q(D_1, a_1^1, \theta)$, $Q(D_1, a_1^2, \theta)$,..... , $Q(D_1, a_1^{n_1}, \theta)$ where n_1 is the number of actions or power allocations possible in the first stage. In general, the k^{th} output of the RBF network at i^{th} stage is given by the expression:

$$Q(D_i, a_i^k, \theta) = y_i^k$$

$$= \sum_{j=1}^{j=m_i} W_{Q_i}[k][j] * \phi_i^j(D_i) \quad (5)$$

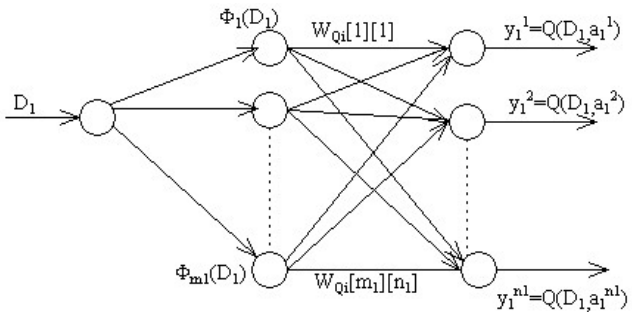

Fig 2 RBF NETWORK FOR STAGE 1.

where W_{Q_i} is n_i x m_i matrix of reals, and $W_{Q_i}[k][1]$,..... $W_{Q_i}[k][m_i]$ are the weights connected to the k^{th} output unit; and $\{\phi_i^j, j = 1,.....m_i\}$ is a set of m_i radial basis functions which constitute the m_i hidden units at i^{th} stage.

We use the Gaussian function as the RBF. Then the output of j^{th} RBF (ie, j^{th} hidden unit) is given by,

$$\phi_i^j(D_i) = exp((D_i - c_i^j)^2 / 2\sigma_i^2) \quad (6)$$

where c_i^j is the centre of j^{th} RBF at i^{th} stage and σ_i is the width of the RBFs at i^{th} stage. It may be noted that centers of all m_i RBFs at a particular stage is chosen to be the same. Substituting $\phi_i^j(D_i)$ from eqn (6) to equation (5) we get,

$$Q(D_i, a_i^k, \theta) = y_i^k$$
$$= \sum_{j=1}^{j=m_i} W_{Q_i}[k][j] \, exp(D_i - c_i^j)^2 / 2\sigma_i^2)$$
$$\text{where } k = 1,....n_i \quad (7)$$

The RBF network at i^{th} stage as described above is thus completely specified by the parameter vector

$\theta = [c_i, \sigma_i, W_{Qi}]$

Where $c_i = [c_i^1, c_i^2,.....c_i^{m_i}]$

$W_{Qi} = \{W_{Qi}[k][j], k=1,.....n_i, j=1,....m_i\}$

Thus, finding a good approximation of the Q value function using the RBF network involves finding the optimum parameter vector $\theta^* = [c^*, \sigma^*, W^*]$.

VI. LEARNING ALGORITHM FOR ECONOMIC DISPATCH

In the previous section we concluded that finding the optimum parameter vector $\theta^* = [c^*, \sigma^*, W^*]$ is the task to be resolved in finding the good approximation of the Q value function. Now we explain how to find the optimal parameters, c^*, σ^*, and W^*.

In the RBF network described above, adjusting the weights associated with a given basis function, say j^{th} basis function at i^{th} stage , will effect the value of the output only in a small region around the centre of j^{th} RBF, c_i^j . (For a value of input $D_i \in \chi_i$, away from c_i^j, $exp((D_i - c_i^j)^2/2\sigma_i^2)$ will be almost zero). This feature of RBF network structure which indicates that each hidden layer neuron can be viewed as approximating the target function over a small region around its centre makes it possible to place the centres on a uniform grid spacing in order to get a better interpolation of the input.

In the case of Economic Dispatch problem, we fix the number of hidden units based on the generation limit constraints. The input to this i^{th} RBF network (D_i) is the fraction of the demanded power yet to be distributed among the remaining $N–i$ units. Therefore, number of hidden units in each RBF network depends upon the maximum and minimum possible demand at that particular stage. i.e., for i^{th} stage, it depends on the values of D_i^{min} and D_i^{max} , where D_i^{min} is the minimum load demand possible to be met with $N – i$ units yet to be allocated and D_i^{max} is the maximum value possible. Also at each stage, the input state D_i is decided by the actions or allocations made at the previous stages.

For the first stage or first RBF network, the input will be equal to the power demand 'P_T'.

i.e., $D_1 = P_T$

Minimum and maximum ranges of demand power possible are given by equation (4). The number of hidden nodes is decided by the width $D_i^{max} – D_i^{min}$. We take the m_i centres to be uniformly distributed in the interval between D_i^{min} and D_i^{max}. For the later stages the range $D_i^{max} – D_i^{min}$ reduces. Therefore as i approaches N, number of centres (and thus hidden nodes) can be reduced.

Once the number of hidden nodes is fixed, we find the distance between centres of Gaussian functions as,

$g^i = (D_i^{max} - D_i^{min}) / (m^i -1)$

m^i – Number of RBF centres at i^{th} stage.

Then the optimal values for the m^i centres of the i^{th} RBF network (c_i^*) are chosen uniformly distributed between D_i^{min} and D_i^{max} as

$\{ D_i^{min}, \ D_i^{min} + g^i , ... D_i^{max} \}$

Next is to find the optimum values for the width of the RBF networks at the different stages. Since the centres of the Gaussian functions form a uniform grid, width of the Gaussian function (σ) is taken as a suitable multiple (spread factor) of the distance between the centres. That is, σ= spread factor * distance between centres. The spread factor can be chosen in the range 0.5 – 1.5 based on the complexity of the problem. Value of 0.7 is found to be sufficient for most problems.

Since we had fixed the values for c_i^* and σ_i^*, our task of learning now reduces to finding the optimum value for the third element in the optimum parameterised vector θ^*, W_{Qi}^*. In other words, problem reduces to learning the weights of the Neural Network at each stage, $W_{Qi}[k][j]$, $k=1,....n_i$, $j=1,....m_i$.

We make the network to learn the optimal values for the weight elements $W_{Qi}[k][j]$ such that y_i^k would be a good approximation of $Q^*(D_i, a_i^k)$. If we know the Q^* values for each stage corresponding to a large number of state action pairs, we can go for supervised learning of the weight matrix W_Q^*. But the Q values are unknown and therefore we cannot proceed in this direction. Hence we employ reinforcement learning algorithm for training the RBF network.

In the standard Q learning method [8], updating of Q value at each iteration 'n' is given by the equation:

$$Q^{n+1}(x_i, a^k) = Q^n(x_i, a^k) + \alpha(g(x_i, a^k, x_{i+1}) + \gamma min_{a' \in A^{i+1}} Q^{n+1}(x_i, a') - Q^n(x_i, a^k))$$

Here γ is the discount factor which accounts for future effects and α is the step size of learning.

In this context, for learning the weight vector elements, we make use of Q learning method for updating the weights. It may be recalled that in our architecture, there are N RBF networks. The update equation for W_{Qi} for $i = 1, ..., N-1$ and $i = N$ are different. The two sets of update equations are given below.

Update equation for W_{Qi}, i = 1, ..., N-1

In the network, if the current state is D_i and the action is a_i^k, then we want the updating to be localized to the output node $y_i^k = Q(D_i, a_i^k)$ or in other words the weight values connected to y_i^k need only be modified while keeping the remaining weight values unchanged.

Since $y_i^k = \sum_{j=1}^{j=m_i} W_{Qi}[k][j] \, \emptyset_j(D_i)$, to change y_i^k we need to modify $W_{Qi}[k][j]$, $j = 1,.....m_i$.

Also $W_{Qi}[l][j]$, $l=1,.....n_i$, $l \neq k$ should remain unchanged.

That is, $W_{Qi}^{(n+1)}[k][j] = W_{Qi}^{(n)}[k][j] + \Delta W_{Qi}^{(n)}[k][j]$, $j=1,.....m_i$

$W_{Qi}^{(n+1)}[l][j \ W_{Qi}^{(n)}[l][j]$ $l = 1,...n_i$ *and* $j=1,.....m_i$, $l \neq k$. (8)

Now let us find $\Delta W_{Qi}^{(n))}[k][j]$. Our aim is to minimize the error in the output ($y_i^{k*} - y_i^k$), and in the n^{th} iteration we have an estimate of this error given by

$$g(D_i, a_i^k, D_{i+1}) + \gamma min_{a' \in A_{i+1}} Q_{i+1}^n (D_{i+1}, a', W_{Qi}^{(n)}) - Q^{(n)}(D_i, a_k, W_{Qi}^{(n)}).$$

Hence, $\Delta W_{Qi}{}^{(n)}[k][j]$ is given by:

$$\Delta W_{Qi}^{(n)}[k][j] = \alpha [g(D_i, a_i^k, D_{i+1}) + \gamma min_{a' \in A_{i+1}} Q_{i+1}^n (D_{i+1}, a', W_{Qi}^{(n)}) - Q^{(n)}(D_i, a_k, W_{Qi}^{(n)})] \emptyset_j(D_i) \quad (9)$$

Update equation for W_{QN}

For the last stage, since we have only a single unit to schedule, there is no choice of action but to allocate the power D_N itself, action $P_N = a_N^k = D_N$. Then the *look ahead* term $Q_{i+1}^n (D_{i+1}, a', W_{Qi}^{(n)})$ is zero in the update equation. Therefore update equation is expressed as:

$W_{QN}^{(n+1)}[k][j]) = W_{QN}^{(n))}[k][j] + \Delta W_{QN}^{(n)}[k][j]$,

where,

$$\Delta W_{QN}^{(n)}[k][j] = \alpha [g(D_N, a_i^k, D_{N+1}) - Q^{(n)}(D_N, a_N^k, W_{Qi}^{(n)})] \emptyset_j(D_N) \quad (10)$$

At each step of the iteration, the weight values are updated and as learning progresses, Q values will approach to Q^*. The selection of Gaussian distribution function as the functions defining the hidden nodes, make the updating a localized task which in turn improves the computational efficiency.

Once the learning is completed, we get the allocation schedule by just retrieving the action element at each stage which corresponds to minimum Q values (greedy action). The entire algorithm for finding the optimum allocation schedule is having two phases: learning phase and policy retrieval phase and are given in algorithms I & II

ALGORITHM I
LEARNING PHASE

Read the parameters of the RL algorithm
Read the Generating unit details
For each of the N stages
Do
Find D^{min} *and* D^{max}
Fix suitable number of hidden nodes for each RBF network
Find the different centres of the RBF network
Choose a suitable action step S_a
Find the set of actions at each stage and thus number of output nodes for each network
Initialize all the weight matrix elements as zero
End Do

For iteration =1 to max_iteration
DO
P_T *= rand* (D_1^{min}, D_1^{max})
Initialize state of the system, $D_1 = P_T$
For i =1 to N-1
Do
Find the permissible action sub set $\mathcal{A}_i$
Select an action a_i^k *from the permissible action set* $\mathcal{A}_i$ *using* $\in$ *- greedy strategy*
Find the allocation for i^{th} *unit,* $P_i = a_i^k$
Find the cost of generation C_i (P_i^k)
Update the weight vector elements connected to y_i^k *using equation (8)& (9)*
Update the state, $D_{i+1} = D_i - P_i$
End Do
For stage =N
$a_N^k = D_N$
Update the weight vector elements connected to y_N^k *using equation (10)*
END DO

ALGORITHM II
RETRIEVING POLICY

Read the load demand value D
Intial demand D_1 *=D*
For i = 1 to N
Do
Compute the Q values corresponding to different actions
Find the action with minimum Q value, argmin$Q(D_i, a_i^k)$
Allocation $P_i = a_i^k$
$D_{i+1} = D_i - P_i$
End Do

VII. LEARNING ALGORITHM WITH TRANSMISSION LOSSES CONSIDERED

Now we extend our previous algorithm in order to incorporate the losses in the transmission system. We use the B matrix loss formula for calculating the loss MW at each step. We find the dispatch of the different possible load values considering transmission losses as described below.

We execute Algorithm I to learn the Q values. The optimum generation schedule corresponding to a load demand $D_1 = P_T$ is obtained by running the algorithm II. Then from the optimum allocation obtained, corresponding loss is calculated using B coefficients. The demanded power is then modified as $D_1 = P_T$ + Loss. Then the dispatch is again obtained for the power D_1 using algorithm II. The loss is again calculated and the change in loss between two successive iterations is found out. This is continued until the difference in loss between two successive iterations become negligibly small. The above procedure is continued for all the load values in suitable steps.

VIII. SIMULATION AND RESULTS

In this section, simulation results of the constrained Economic Dispatch problem of IEEE six bus system are demonstrated. The algorithm is applied to obtain the schedule for several load values. A comparison is carried with other recent technique.

The fuel cost curve of the units is represented by a third order polynomial function [5]. The associated fuel cost coefficients and B-matrix parameters are given in Table I. We find the dispatch for various load values ranging from minimum demand possible to maximum value possible.

In order to apply the new algorithm, first we should fix the parameters of the RBF networks. One important parameter we should fix is the number of centres. As mentioned in the previous section, number of centres depends on the input space or the range between D_i^{min} *and* D_i^{max}. We calculate the range of demand input to each stage as

$D_1^{min} = 400$ $D_2^{min} = 300$ $D_3^{min} = 200$
$D_1^{max}. = 2000$ $D_2^{max}. = 1500$ $D_3^{max}. = 1000$

Since the range of input power to the different RBF networks are different, for increasing the computational efficiency, we select more number of centres at the initial stages compared to later ones. Here we choose the number of centres as 80, 60 and 40. Once we fix the number of centres, to get the width of the RBF networks, a suitable spread factor is to be selected. Spread factor decide the percentage of overlapping between two successive functions. By trial and error spread factor of 0.7 is selected in all the RBF networks. Next is to fix the output nodes. For this, a suitable action step S_a is to be selected. In our simulation we choose 2 MW as the step size in order to get accuracy when transmission losses are considered. Then the umber of actions and hence number of output nodes at each stage are calculated as,

$$n_i = \frac{\left(P_i^{max} - P_i^{min}\right)}{S_a}, \qquad i = 1,..N$$

Next, the Learning parameters of the Reinforcement Learning algorithm are decided. In case of $\in$ - greedy action selection strategy, a larger value of $\in$ increases the exploration rate in the action space, while smaller rate increases the extent of exploitation. Therefore, initially a value of 0.5 is taken for $\in$ providing sufficient exploration in the initial step. As the learning process proceeds, greedy action is selected with increased probability by reducing value of $\in$ gradually.

Discount parameter γ accounts for the discount to be made in the present state for accounting of future reinforcements and since in the case of Economic Dispatch problem, the cost

of future stages has the same implication as the cost of the current stage, value of γ is taken as 1. The step size of learning is given by the parameter α and it affects the rate of modification of the estimate of Q-value at each iteration step. By trial and error α is taken as 0.1 for achieving sufficiently good convergence of the learning system.

We find the dispatch for various load values ranging from minimum demand possible to maximum value possible. In this simulation, since the discretisation step for action set is 2MW, there could be a maximum difference of 2MW between power generation and power demand including losses. This remaining power of the demanded value (less than 2MW), which is negligibly small compared to the total demand is randomly assigned to one of the units without exceeding maximum limit. Learning phase using Algorithm I converged in 10^5 iterations. Then the dispatch for 6 values of load demand is found. The total time taken is only 6.98 sec. The schedule and the total cost are tabulated in Table II. The other stochastic methods like Simulated Annealing, Genetic Algorithm etc. require learning to be carried out for each load value. This indeed need 6 times the time required for learning a single load demand.

While comparing the schedule for a load of 1200MW, the cost and load distribution are found to be almost same as those obtained through simulated annealing [4] and RLED [12]. For obtaining the schedule for this single load value, simulated annealing technique had taken 27.25 sec while the proposed algorithm took only 6.9 sec for giving schedule of all load values. RLED took a time of 18.68 sec with a discretisation step of 2MW. Thus the proposed algorithm seems to be better than these strategies.

For validating the proposed algorithm for the wide range of load values, we obtained the schedule for 24 hour load pattern of the IEEE 6 bus system with three generating units. The schedule of allocation is obtained in 6.98 sec. Also the proposed algorithm was validated for IEEE 30 bus system with six generating units. Due to the space limitation we are not including the results of these here.

IX. CONCLUSION

In this work, we developed a function approximation approach for solving Economic Dispatch problem. Here, we combined the Reinforcement Learning with the function approximation capability of Neural Networks. The function approximation capability of Neural Networks gives out the advantage of the input state space to be continuous instead of discrete. Also once the learning phase is completed, we can get the schedule for any load demand instantaneously. We have tested the efficacy of the algorithm for the IEEE six bus system with third order cost functions and also considering transmission losses. Also from the comparison with other methods, we see that the new algorithm gives the optimum allocation with comparable cost, with much lesser time as compared to other methods. Also since with a single learning task, it is easy to retrieve schedule for any demand in the continuous range, it is more suitable for practical systems.

Due to the localized updating of the weight vector elements, the proposed algorithm seems to be very much promising where the number of units and hence the range of demand power at each stage extends more.

TABLE I
COST COEFFICIENTS AND B – COEFFICIENTS

Ca	Cb	Cc	Cd	Pmin	Pmax
11.20	5.10238	-0.0026429	3.33333e-6	100	500
632	13.01	-3.05714e-2	3.3333e-5	100	500
147.14	4.28997	3.0845e-4	-1.7677e-7	200	1000
B coefficients:					
7.5e-5		5e-6	7.5e-6		
5e-6		1.5e-6	1e-5		
7.5e-6		1.0e-5	4.5e-5		

TABLE II
COST CHARACTERISTICS

D (MW)	P_1	P_2	P_3	Cost	Loss(MW)
521	100	100	328	2393	7
713	274	100	354	3257	15
876	328	100	470	3756	22
1098	339	100	691	5200	32
1200	343	100	800	5679	43
1460	453	229	832	6852	54

REFERENCES

[1] A.J.Wood, B.F.Woolenberg. *Power Generation and Control.* John Wiley Sons 2002.

[2] C.L. Chen, S.C. Wang. *Branch and bound scheduling for thermal generating units,* IEEE Trans. Energy Convers. , 8 (1993) : 184 – 189

[3] P.H. Chen, H.C. Chang. *Large Scale Economic Dispatch by Genetic Algorithm,* IEEE Transactions on Power Systems 10, 4 (1995) : 1919 – 1926.

[4] K.P.Wong, C.C. Fung. *Simulated Annealing based Economic Dispatch algorithm*, IEE Proc.- C 140, 6 (1993) : 509 – 515.

[5] Whei Min Lin, Hong – Jey Gow. *A Partition Approach algorithm for nonconvex Economic Dispatch,* Electric Power and Energy Systems (2007).

[6] T.Jayabarathi, K. Jayaprakash.*Evolutionary Programming techniques for different kinds of Economic Dispatch problems,* Electric Power Systems Research 73 (2005) : 169 – 176.

[7] D.P.Bertsekas, J.N.Tsitsikilis. *Neurodynamic Programming,* Athena Scientific, Belmount,MA., 1996.

[8] R.S.Sutton, A.G.Barto. *Reinforcement Learning: An Introduction,* Cambridge, MA:MIT Press, 1998

[9] T.P.Imthias Ahamed. *A Reinforcement Learning Approach to Unit Commitment Problem*, Proceedings of National Power System Conference 2006

[10] G.R.Gajjar, S.A. Khaparde, P.Nagaraju, S.A.Soman. *Application of Actor Critic Learning algorithm for optimal bidding problem of a Genco,* IEEE Transactions on Power Systems 18, 1(2003) : 11 – 18.

[11] Vishuteja Nanduri, Tapas K.das, *A Reinforcement Learning Model to assess Market Power under auction based strategy*, IEEE transactions on Power Systems 22 , 1 (2007) : 85 - 95

[12] E.A.Jasmin, T.P.Imthias Ahamed, V.P.Jagathiraj. *A Reinforcement Learning Algorithm for Economic Dispatch considering transmission losses,* Accepted for TENCON 2008, IEEE region 10 conference.

[13] S.Haykin, *Neural Networks : A Comprehensive Foundation.* PHI 1999

[14] T.P.ImthiasAhamed, P.S.NagendraRao, P.S.Sastry. *A Neural Network based Automatic generation controller design through Reinforcement Learning,* International journal of Emerging Electric Power Systems 2006; 6 (1) .